PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 9/40 — A1

(11) International Publication Number: WO 98/04971

(43) International Publication Date: 5 February 1998 (05.02.98)

(21) International Application Number: PCT/US97/12214

(22) International Filing Date: 25 July 1997 (25.07.97)

(30) Priority Data:
08/686,312 25 July 1996 (25.07.96) US

(71) Applicant: TRADEWAVE CORPORATION [US/US]; Suite 100, 3636 Executive Center Drive, Austin, TX 78731 (US).

(72) Inventors: PAINTER, Paul, B.; 8812 Grape Cove, Austin, TX 78717 (US). HARDIN, John, W.; 8702 Pepper Rock Drive, Austin, TX 78717 (US).

(74) Agents: SHOWALTER, Donald, S. et al.; Holland & Knight LLP, One East Broward Boulevard (33301), P.O. Box 14070, Fort Lauderdale, FL 33302-4070 (US).

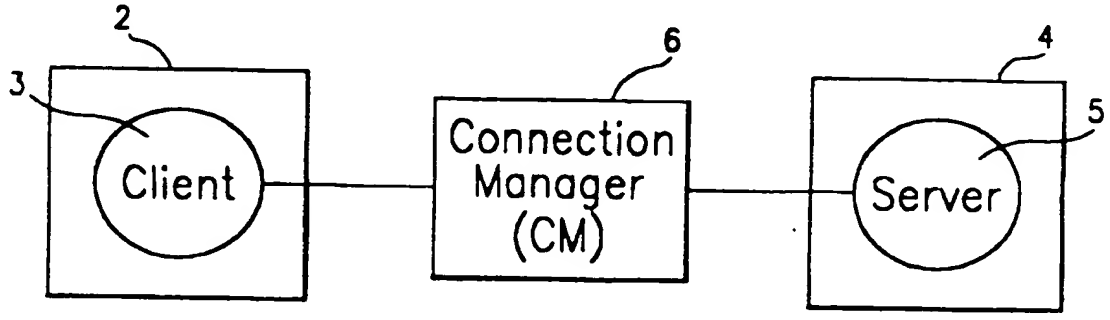
(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, HU, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, TJ, TM, TR, TT, UA, UG, UZ, VN, ARIPO patent (GH, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, ML, MR, NE, SN, TD, TG)

Published

With international search report.

Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: METHOD AND SYSTEM FOR GENERALIZED PROTOCOL IMPLEMENTATION ON CLIENT/SERVER COMMUNICATIONS CONNECTIONS



(57) Abstract

A client (3) and server (5) interposed by a connection manager (6) for implementing communications protocols between a client (3) and server (5) in a transparent, application-independent, non-invasive fashion. The connection manager (6) comprises a client component (3) typically resident on the client machine (2) and a server component (5) typically resident on the server machine (4). The client component (3) accepts connection requests from client applications and sets up those connections in cooperation with the server component (5) of the connection manager. The connection manager (6) identifies the type of connection requested (e.g., ftp, http, gss-http) based on such things as the content of the request and invokes methods specific to the type of connection requested. In this manner, the connection manager carries out the higher-level protocols, such as security protocols, for the connection in a way that is transparent to both the client (3) and server (5).

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

INTERNATIONAL SEARCH REPORT

International application No
PCT/US97/12214

A. CLASSIFICATION OF SUBJECT MATTER

IPC(6) :G06F 9/40
US CL :395/680

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/680; 683, 187.01, 186, 188.01, 200.1

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

APS

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No
Y	US 5,506,961 A (CARLSON et al) 09 APRIL 1996, col. 7, line 12, col. 8, fig. 3.	1-2
Y	US 5,509,121 A (NAKATA et al) 16 APRIL 1996, col. 1, lines 30-35, col. 2, lines 38-40, col. 3, lines 55-63.	1-2

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 08 OCTOBER 1997	Date of mailing of the international search report 12 DEC 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer KEVIN KREISS Telephone No. (703) 305-9600

Form PCT/ISA/210 (second sheet)(July 1992)*

E. CM/S: Write Client Request to Server
IV. Client: Read Response from Server (CM/C)
A. CM/S: Read Server Response
1. CM/S: Read Server Response Line
HTTP/1.0 200 OK[CR/LF]
2. CM/S: Read Server Headers and Body
Content-type: text/plain[CR/LF]
[CR/LF]
Hello, John Smith!
B. CM/S: Write Server Response to Client
C. CM/S: Close Server Connection
D. CM/C: Read Server Response
1. CM/C: Read Server Response Line
2. CM/C: Read Server Headers and Data
E. CM/C: Write Server Response to Client
F. CM/C: Close Server Connection
V. Client: Read and Display Server Response
VI. Client: Close Connection to Server (CM/C)

FIG. 5F

METHOD AND SYSTEM FOR GENERALIZED PROTOCOL IMPLEMENTATION ON CLIENT/SERVER COMMUNICATIONS CONNECTIONS

FIELD OF THE INVENTION

The present invention relates to computer networks having enhanced and/or extended client/server communications. More particularly, the present invention is characterized by an application-independent, object-oriented connection manager for processing client/server connections (communications sessions) between client computers and server computers.

BACKGROUND OF THE INVENTION

Client/server communications between client computers and server computers are commonly established by the interaction of an application program running on the client and a corresponding application program running on the server. The client/server model is the conventional model governing the transfer of data between application programs on a computer network. According to this model, the high-level protocols for reading and writing data between a first computer (the client) and a second computer (the server) are embedded in the application software running on the client and server, respectively. Prior to the transfer of data, a communications session must be established over the network between the client and the server.

Such a communication session is established according to a number of "layers" of protocols. Among the lowest level protocols, physical connectivity between the client machine and the server machine is established and maintained. For example, the Ethernet CSMA/CD protocol is a common data link layer protocol governing the orderly

transmission of packets of data between a client and server. Higher-level protocols, such as the TCP/IP and XNS transport layer protocols, govern the assembly of data into messages and the uniform addressing of various computers on the network. Due to the established nature of protocols at these levels, much client software has these protocols "built-in" so that these protocols are automatically employed when the client software is run on a client or server machine on the network.

Still higher level protocols govern the interoperability of particular client/server applications such as file transfer, remote file access, electronic mail, etc. Examples of such applications include Internet-based applications, where use of a file transfer protocol permits the transfer of files from ftp server sites (ftp://xxx.xxx); a different protocol permits a client to browse documents in hypertext mark-up language (html) format at http server sites (http://xxx.xxx); and yet another protocol governs the establishment and maintenance of secure communications sessions between a client and server at gss-http sites (http://xxx.xxx:488). For these higher-level protocols, each client/server application is typically specially adapted at the source code level to implement these protocols. In other words, the task of setting up a client/server communications session employing the desired protocol is typically accomplished by application-specific code developed by the application vendor for permitting the application program to be run on a client and communicate with a server on the network using higher-level protocols such as ftp, http, or gss-http.

Protocols for establishing secure client/server connections have conventionally been handled in the foregoing manner. For example, Internet Web-based client applications often employ software security packages to establish secure client/server communications, but only after the application software has been invasively modified at the source code level to interoperate with the security program. Thus, the application program at the server must "know" that a security protocol is being employed, and the application program must be specially adapted to work with the security protocol.

The need for transparent and application-independent client/server communications management for implementing a variety of higher-level communications protocols has been recognized. For example, Gradient's WebCrusader software allows users to securely access distributed applications using standard, off-the-shelf Web browsers installed on desktop client systems. This product is purported to establish a secure session between an off-the-shelf Web client application and a Web server using the DCE (Distributed Computing Environment) RPC (Remote Procedure Calls) protocol. WebCrusader comprises an application-independent "Connect Client" function resident on the client machine which interacts with the client application, usually a Web browser. The Connect Client, in conjunction with a corresponding "Connect Server" function resident on the server, uses the DCE RPCs to forward requests from the Web browser to the server. The Connect Server function receives these requests, performs security checks, fetches the requested document, and uses DCE RPCs to send the document securely back to the Connect

Client for forwarding on to the Web browser. The Connect Client acts as a "proxy," intercepting document requests from the Web browser and determining whether a secure document is sought. If the URL of the requested document contains a DCE name, the Connect Client uses DCE RPCs to forward the request to the Connect Server resident on the secure server. If the URL contains a non-secure address, the Connect Client forwards the request to the appropriate standard Web server using http. In this way, the WebCrusader transparently performs the security functions of authentication, authorization and encryption between a client application and a server using DCE. The software is strictly limited to the RPC protocol between client and server, however, and is not stream-based.

What is needed is an application-independent client/server connection manager suitable for use with any higher-level protocol, such as ftp, http, DCE and gss-http without having to convert to RPC call sequences. It is desired that such a connection manager tansparently set up and manage both secure and non-secure client/server byte-stream sessions, yet inter-operate with each application only at the object code level.

SUMMARY OF THE INVENTION

According to a first aspect of the invention, there is provided a client machine and a server machine in a computer system, wherein the client and server include a connection manager for establishing communications sessions using higher-level

protocols such as http, ftp, or gss-http. The client machine may be any computer capable of running a client application, and will typically include a memory device such as a hard disk drive for storing the client application; a processor for executing the client application; and means for handling input/output (I/O).

The connection manager typically comprises a client component running on the client machine, and a server component running on the server machine. These dual components of the connection manager together manage a series of connections between a client application and a server application. The client component receives requests from the client application and uses the request to set up and manage a communications session with the server application wherein responses from the server are received by the server component of the connection manager. The client aspect and the server aspect are thus in part mirror images of each other, and they function jointly as an "agent" of the communications between the client and the server.

It will be understood that the connection manager, including its client and server components, can be run on a machine other than the client machine (on which the client application runs) or the server machine (on which the server application runs). For example, in a networked office environment, a client machine running a client application may be interoperable with a separate machine running the connection manager, which may then apply the appropriate protocols and enhancements to the connection between the client application and a distant server application.

The connection manager handles requests from a client application using an object-oriented approach to process (set up and manage) the communications session between the client application and the server application. The connection manager is "object-oriented" in that it uses various discriminators determinable from the client or server communication content (e.g., protocol, client or server address, data type, etc.) to evaluate which type of communications connection, or class, is called for. By observing these discriminators, the connection manager can "type" the object and call on the communications methods corresponding to that class of connection when setting up the communications session and carrying out the communications protocols between the client and the server.

According to a second aspect of the invention, the connection manager is application-independent. A variety of client/server applications may be used with the connection manager, and these applications need not be adapted for use with the connection manager. The connection manager receives ordinary requests originated by either the client or the server and uses the content of those requests to set up and manage a communications session between the client and the server. Requests from a wide variety of applications and for a wide variety of classes of connections can be accommodated by the connection manager in this manner.

According to a third aspect of the invention, the connection manager maintains one or more active "listener" objects that await requests for connections of particular types. When a connection is accepted on a particular listener, the connection manager

associates with that connection the group of communications methods for connections of that class.

According to a fourth aspect of the invention, there is provided a client/server connection manager which is non-invasive with respect to the various applications with which it may operate. The connection manager receives high-level, connection-specific requests from the client. The connection manager uses these requests to determine the lower-level protocols required for creating the desired type of connection, such as a secure communications session. The connection manager employs client-resident portions and server-resident portions to manage the communications and supply the lower-level protocols without any modifications to the client application or the server application at the source code level.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a simple block diagram representation of a client/server connection manager according to the present invention.

FIG. 1B is a block diagram of a client/server connection manager separated into its client and server components in a distributed computing environment.

FIG. 2A is a functional design diagram of the client component of a connection manager with active listener objects which determine when connections of various types are requested by a client application.

FIG. 2B is a functional design diagram showing a single listener object associated with a single class of methods for an http connection.

FIG. 3A is a flow chart of the functioning of a client/server connection manager in the computer system according to the present invention.

FIG. 3B is a more detailed textual outline of the flow chart shown in FIG. 3A.

FIG. 4A is a communications activity flow diagram showing the establishment of a communications session between a client and a server using only the client component of a connection manager.

FIG. 4B is a textual outline of the communications activity shown in FIG. 4A.

FIG. 5A is a communications activity flow diagram showing the establishment of a secure communications session between a client and a server using both the client component and a server component of a connection manager.

FIG. 5B is a textual outline of the secure communications activity shown in FIG. 5A.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Referring now to the Figures, the invention in its simplest form is illustrated in FIG. 1A, wherein a client machine 2 and a server machine 4 are interposed by connection manager 6. Client machine 2 may have resident thereon one or more client applications 3, and these applications will typically interact with one or more server applications 5 to obtain data, files, graphics, etc. from the various servers. For

example, server machine 4 may be a server for one or more of ftp, http, or gopher programs on the Internet, or it may be an electronic mail server or other server on a private network. The invention encompasses any type of client/server combination where such is interposed by a connection manager of the type described.

Connection manager 6 in its simplest form resides on client machine 2 and manages the connections between the various client applications 3 and their target server applications 5. Connection manager 6 sets up and manages these client/server connections using an object-oriented approach to determine, based on the type of connection sought by the client application 3 or the server application 5, the appropriate class of communications methods to apply to the connection so that input/output (I/O) between the client and server is processed seamlessly and transparently. The object-oriented approach of connection manager 6 is described more fully below with reference to FIGS. 2A and 2B.

Referring to FIG. 1B, connection manager 6 is shown to comprise client and server components where the client machine 2 and server machine 4 communicate at the application level over a network 8. Those components are designated connection manager/client (CM/C) 10 and connection manager/server (CM/S) 12, respectively. CM/C 10 typically receives from the various client applications 3 certain connection requests destined for server applications 5. CM/C 10 types the object, or connection sought, according to the content of the request from client application 3. CM/C 10 and CM/S 12 then cooperate to establish the requested type of connection, applying the

appropriate higher level protocols through the collection of methods specific to the requested type of connection. For example, if client application 3 requests a secure (gss-http) Web connection, CM/C 10 and CM/S 12 invoke the communications methods associated with the gss-http class of connection; neither the client application 3 nor the server application 5 may be aware that the gss-http protocols are being implemented by CM/C 10 and CM/S 12.

Once a connection is established by the components of connection manager 6, I/O between client application 3 and server machine 4 is dispatched by CM/C 10 and CM/S 12 in a way that is transparent to the client and server. CM/C 10 invokes the communications methods associated with the particular type of connection established (e.g., Init, connect, read and write methods for an http-class connection), thereby to carry out the necessary protocol and giving the appearance that client application 3 is receiving I/O from the server application unaided by any other application or utility. Thus, it can be seen that CM/C 10 interacts with client application 3 as if CM/C 10 were in fact a server application. Likewise, server application 5 interacts with CM/S 12 in the same way that server application 5 would ordinarily interact with a client application. The connection is thus transparent to client application 3 and server application 5, which require no special knowledge that the components of connection manager 6 are interposed between the client and the server.

Referring now to FIG. 2A, the establishment of client/server connections by CM/C 10 is described. FIG. 2A representationally shows a plurality of logical

connections established between various client applications 3' and 3" and their target server applications via a computer network. CM/C 10 is preferably provided with an intializer 30 for setting up a plurality of listener objects 16. Listener objects 16 are tailored, through their association with connection classes 31, to detect a request for a particular type of connection from client applications 3' and 3", wherein each connection class 31 defines a collection of communications methods that implement protocols specific to a connection type. When CM/C 10 accepts a connection from a client application 3' or 3", CM/C 10 associates with that connection the communications methods corresponding to that type of connection.

When CM/C 10 accepts a connection request from client application 3' or 3", CM/C 10 creates a connection object 24 for http connection 18, a connection object 26 for http connection 20, or connection object 28 for gss-http connection 22. Connection objects 24, 26 and 28 are specialized by their respective connection classes 31 to handle connections of particular types. The connection objects have pointers to the actual client and server connections, pointers to specific methods of connection classes 31, and other information allowing them to be managed by CM/C 10. For example, the http connection class's Init method (http_Init) sets connection object 24's read, write, exception, and deinit methods to http_Read, http_Write, http_Exception, and http_Deinit, respectively. Event manager 14 calls the connection class methods for each connection object in order to process I/O events and apply enhancements to the protocols for each connection, e.g., adding security to http.

CM/S **12** is preferably designed to mirror the operation of CM/C **10** and may be designed similarly to accept communications requests on behalf of server application **5** and receive communications from the server. For simple connections such as http connection **18**, where data is written to and from CM/C **10** without the need for enhanced protocols such as security protocols, CM/S **12** may not be active on the server side, and connection manager **6** may then be considered to comprise only CM/C **10**, which connects directly to server application **5**.

FIG. **2A** shows the establishment of three different types of connections by CM/C **10**, although any type of connection can be accommodated by the present invention. There is shown representationally a logical, non-secure http connection **18** between a client application **3'** and a server application **5** (not shown). This connection typifies the non-secure Internet communications between a client and server. Also shown are http connection **20** and secure gss-http connection **22**, which employs the gss-http protocol. CM/C **10** cooperates with CM/S **12** (not shown) to set up each connection on appropriate ports of server machine **4**. For example, http connection **18** and http connection **20** are typically set up on non-secure ports 80, while secure gss-http connection **22** is set up on secure server port 488 according to current standards.

After the connections are established by CM/C **10** and CM/S **12**, I/O between client application **3** and server application **5** is processed according to a collection of methods which are specific to the type of connection established. In FIG. **2A** are shown connection objects **24**, **26**, and **28** which are associated with the methods for the http

class, http class, and gss-http class of connection, respectively. Typical methods include read, write, init and connect, although numerous methods may be invoked depending on the connection type and the methods which are suitable for the particular connection type. The functioning of current methods is well known. For example, the http read method is invoked any time input data from either the client application 3 or the server application 5 is available on an associated connection. The http init method is invoked as soon as a client connection is accepted by CM/C 10. And the http connect method is invoked as soon as a server connection is established by either CM/C 10 or CM/S 12.

The methods of FIG. 2A may be implemented in any of a variety of ways, including subroutines (both statically and dynamically linked), executing local applications, remote procedure calls, Active-X and Java. Statically and dynamically linked subroutines have proven to be an acceptable means for implementing these methods.

Referring now to FIG. 2B, there is shown a functional design diagram of CM/C 10 for a single http-class connection. Connection class 31' defines the class of http methods specific to the http connection class. Listener object 16' receives a connection request on http connection 18. In response, CM/C 10 accepts the connection request and creates connection object 24, which has pointers to the http read method 60', write method 66', connect method 64' and exception method 65. CM/C 10 also associates connection object 24 with the http methods of connection class 31'. Event manager 14

makes calls to these methods to process the I/O on http connection **18**. I/O events (e.g., reads, writes, and exceptions) may originate from either the client or the server. Processing of these read and write events is handled by event manager **14** via process read routine **48** and process write routine **50**, which make the appropriate calls to the methods associated with the connection object.

Referring now to FIG. 3A, there is shown a flow chart of the functioning of connection manager **6**, which preferably has basic functions provided by an initialization routine **30**, an event processing routine **32**, and a quit routine **34**. Initialization routine **30** is executed once upon startup, and includes steps preliminary to establishing client/server connections. In step **36**, initializations specific to the platform on which connection manager **6** is installed are carried out. The various program modules of connection manager **6** are initialized in step **38**. The connection class definitions (reference numeral **31** in FIG. 2A) for connections to be managed by connection manager **6** are loaded in step **40**. The appropriate methods for use with each connection class are initialized in step **42** and the listener objects **16** (FIG. 2A) are created in this step.

Event processing routine **32** processes I/O events, and these are preferably processed according to a process read step **48**, a process write step **50**, and a process exception step **52** for handling error conditions on the connection. In step **54**, a read request is examined to determine whether it originated from the client or the server. If the read request originated from the client, step **56** determines whether it is a request to

read data from the client or instead a request to accept a connection from the client. Requests to accept a connection invoke Init method **58** to specialize the connection object **24** created by connection manager **6**. In contrast, requests to read data invoke read method **60**. Read requests originating from the server likewise invoke read method **60**, which occurs when data is to be read from the server and written to the client.

Write requests are handled by process write step **50**, which first determines (step **60**) whether the request is destined for the client or the server. If the write request is destined for the server, step **63** determines whether it is a request to write data to the server or instead a signal that a server connection request previously issued has completed. Requests to write data invoke write method **66**. Write requests destined for the client likewise invoke write method **66**, which occurs when data is to be written to the client.

Event processing routine **32** may handle events other than I/O events. These include user events, such as an indication from the user that he wants connection manager **6** to exit. They may also include programmatic events (e.g., another program wants connection manager **6** to exit), and so on.

In the typical http connection, the read, write, Init and connect methods are sufficient to establish the communications session and handle all I/O between a client application **3** and a server machine **4**. If communication is over a network, components CM/C **10** and CM/S **12** typically run respectively on client machine **2** and server

machine **4**, with each component handling both read and write requests from the client and the server according to the steps above.

When all connections are to be terminated, quit routine **34** carries out step **68** to delete any active connection objects (which closes all open connections) and step **70** to deinitialize the classes of connections previously initialized in step **40**. FIG. **3B** shows in outline form the various steps of FIG. **3A**.

Example A: Unsecure Web Connection (http)

Referring to FIG. **4A**, there is shown an activity flow diagram for a simple, non-secure client/server connection which is set up and managed by connection manager **6**. In the example, an http class connection is established on the Internet; client application **3** is therefore presumed to be a Web browser and server machine **4** is presumed to be a Web server. Because no higher-level network protocols (such as security protocols) are required, connection manager **6** resides only on the client machine **2**. Prior to the initiation of the activities shown in FIG. **4A**, it is presumed that connection manager **6** has been initialized and that at least one listener object **16** (FIG. **2**) of the http class is active.

The operation of connection manager **6** for http-class connections is now described. Typically, a user of client application **3** (Web browser) enters the Universal Resource Locator (URL), such as:

http://server.com

which identifies a Web site to be browsed (step **101**). In response to the user entry of a

URL, client application 3 in step 102 attempts to open a connection to CM/C 10, which client application 3 treats as a server application 5. Connection manager 6, which has at least one active http listener object 16' (FIG. 2A), accepts the connection and invokes the Init method associated with the accepting connection object (in this case, http_init). At this point and throughout the example, client application 3 interacts with connection manager 6 as if it were the server; thus client application 3 now treats the connection as having been established with server application 5. In step 104, client application 3 writes an http request, such as:

GET http://server.com/HTTP/1.0[CR.LF]
[CR/LF]

to connection manager 6. In step 105, connection manager 6 reads this request using the read method for http. After successfully reading the request line, the http class parses the specified URL to determine if it is valid. If the URL is not valid, then connection manager 6 signals an error. After successfully parsing the specified URL, the http class read method next reads the http header lines.

After the header lines are read, connection manager 6 cooperates with the server to establish a connection using the connect methods (steps 106-108), and subsequently invokes the http write methods to write the client request to server machine 4 (step 109). Once the server reads this client request (step 110), it writes a response (step 111). According to the http protocol, this response consists of a status line, a header, and the body of the response, e.g.:

HTTP/1.0 200 OK[CR/LF]
[CR/LF]
Hello World!

This response is then read by connection manager **6** (step **112**). Like the client application, the server's communication through connection manager **6** is transparent, and the server cannot discern that connection manager **6** is anything other than an http client.

Once connection manager **6** has received the response from the server, it writes the response to the client application **3** (step **113**). Client application **3** then reads the response and displays it to the user (step **114**). Thereafter, connection manager **6** closes the server connection in step **115**, and subsequently client application **3** closes the connection with connection manager **6** in step **116**. The steps associated with all of the activities in this http example of FIG. **4A** are shown in outline form in FIG. **4B**.

The particular methods implementing the http class connection in Example A are outlined generally as follows:

I. http_ClassInit
    A. CreateListener CO
    B. ConnObj_SetInitMethod(http_Init)
    C. CreatAdmin?
        1. CreateListener
        2. ConnObj_SetInitMethod(http_Init)
II. http_Init
    A. ConnObj_SetReadMethod(http_Read)
    B. ConnObj_SetWriteMethod(http_Write)
    C. [ConnectionManager/Server]
        1. CallMethod(gss_Init)
III. http_DeInit
IV. http_Connect

V. http_Read
A. ReadRequestOrReadResponse?
1. http_Read_Client_Request
a) http_Read_ClientInit
b) http_Read_Request
c) http_Read_ClientHeaderAndBody
(1) LocalOrRemote?
(a) http_ProcessLocalURL
(b) http_ProcessRemoteURL
i) SecureOrUnsecure?
(1) SetConnectMethod(gss_Connect)
(2) SetConnectMethod(http_Connect)
ii) OpenRemoteConnection
iii) Buffer/FlushRequest
d) http_Read_ClientDone
2. http_Read_Server_Response
a) http_Read_ServerInit
b) http_Read_Status
c) http_Read_ServerHeaderAndBody
d) http_Read_ServerDone
VI. http_Write
A. WriteRequestOrWriteResponse?
1. http_Write_ClientRequest
a) WriteComplete?
(1) SwitchToRead/WriteResponse
2. http_Write_ServerResponse
a) WriteComplete?
(1) ConnObj_MarkForDeletion
VII. http_Exception

Example B: Secure Web Connection (gss-http)

Referring to FIG. **5A**, there is shown an activity flow diagram for a secure gss-http client/server connection which is set up and managed by client and server components of connection manager **6**, namely CM/C **10** and CM/S **12**. As in Example A, client application **3** is presumed to be a Web browser and server machine **4** is presumed to be a Web server. Prior to the initiation of the activities shown in FIG. **5A**, it

is presumed that CM/C **10** and CM/S **12** have been initialized and that at least one listener object **16** (FIG. **2**A) of the http class is active on each of CM/C **10** and CM/S **12**.

The operation of CM/C **10** and CM/S **12** for secure gss-http connections is now described. At the outset, the user of client application **3** (Web browser) gives an indication in step **201** that he desires secure communications with a server. In response, client application **3** attempts to open a connection with CM/C **10** (step **202**) and CM/C **10** accepts the connection (step **203**). The client may then write a request to CM/C **10**, such as:

```
POST http://server.com:488/cgl-bin/foo HTTP/10[CR/LF]
Content-length: 17
Content-type: application/x-www-form-unencoded[CR/LF]
[CR/LF]
name=John%20Smith
```

which includes a request line, header lines, and a user ID. CM/C **10** invokes the read method to read this request in step **205**.

The http read method observes the content of the request and determines that a gss-http secure connection is desired. Thus, the gss connect method will be set as the connect method associated with connection object **24** in FIG. **2**B. Next, the http read method opens a connection to the server (steps **206-208**). CM/C **10** then cooperates with CM/S **12** to establish a connection by invoking the connection object's connect method (in this case, gss_Connect), which performs security context negotiation prior to the transfer of any secure data (steps **209-216**). The server application **5** (Web server)

receives no requests and takes no part in the connection set-up until after CM/C **10** and CM/S **12** have successfully negotiated the secure connection.

Upon completion of the security context negotiation between CM/C **10** and CM/S **12**, http write methods including security protocols are invoked by CM/C **10** to send the client request securely to CM/S **12**. Operating in mirror image fashion, CM/S **12** reads the client request in step **218**. Thereafter, in steps **219** through **225**, CM/S **12** interacts with server application **5** to open a connection to the server, send the client request to the server, and read the server response in a manner analogous to the interaction between connection manager and server application **5** in steps **106-112** in Example A above. The sample response written to CM/S **12** in FIG. **5A** is:

HTTP/1.0 200 OK [CR/LF]
[CR/LF]
Hello John Smith!

Now that the read and write methods invoked by CM/C **10** and CM/S **12** provide for a secure connection, the server response may be securely written to and read by CM/C **10** (steps **226** and **227**) before CM/S **12** closes the server connection (step **228**). After writing the response to the client (step **229**) for display to the user (step **230**), the remainder of the connections are then closed, first by CM/C **10** (step **231**), then by client application **3** (step **232**). Thus, it can be seen from the example that the client and server components of connection manager **6** established a secure gss-http connection between the client and server by interacting with client and server in a way that transparently mimics direct interaction between client and server.

The particular methods implementing the gss class connection in Example B are outlined generally as follows:

I. gss_ClassInit
II. gss_Init
  A. ConnObj_SaveCurrentMethods
  B. ConnObj_SetReadMethod(gss_Read)
III. gss_DeInit
IV. gss_Connect
  A. gss_Init
IV. gss_Read
  A. [ConnectionManager/Client]
    1. gss_Read_Init
      a) gss_AcquireAndFlushToken
    2. gss_Read_Token
      a) gss_Read_TokenHeader
      b) gss_Read_TokenBytes
      c) gss_AcquireAndFlushToken
      d) NegotiationComplete?
        (1) SetSocketParametersForTransparentEncrypt/Decrypt
    3. gss_Read_FirstEncrypted
    4. gss_Read_Done
      a) ConnObj_RestoreCurrentMethods
  B. [ConnectionManager/Server]
    1. gss_Read_Init
    2. gss_Read_Token
      a) gss_Read_TokenHeader
      b) gss_Read_TokenBytes
      c) gss_AcquireAndFlushToken
      d) NegotiationComplete?
        (1) SetSocketParametersForTransparentEncrypt/Decrypt
        (2) ConnObj_SetWriteMethod(gss_Write)
    3. gss_Read_Done
VI. gss_Write
  A. [ConnectionManager/Client]
  B. [ConnectionManager/Server]
    1. BufferAcknowledgement
    2. WriteComplete?
      a) ConnObj_RestoreCurrentMethods
VII. gss_Exception

While a particular embodiment of the invention has been illustrated and described, it will be obvious to those skilled in the art that various changes and modifications may be made without sacrificing the advantages provided by the principles of construction and operation disclosed herein.

CLAIMS

We claim:

1. A computer system providing enhanced communications between a client application and a server application, comprising:

a client machine running said client application;

a connection manager interoperable with said client application to:

(a) receive from said client application a connection request for a specific type of connection;

(b) identify the type of connection requested from a plurality of connection types; and

(c) invoke methods for the type of connection requested, thereby establishing a connection between said client application and said server application.

2. A method of enhancing communications between a client application and a server application in a client/server computing environment, comprising the steps of:

receiving from said client application a connection request for a specific type of connection, said connection request including a body;

identifying the type of connection requested from a plurality of connection types;

invoking methods for the type of connection requested; and

writing said body of the connection request to said server application.

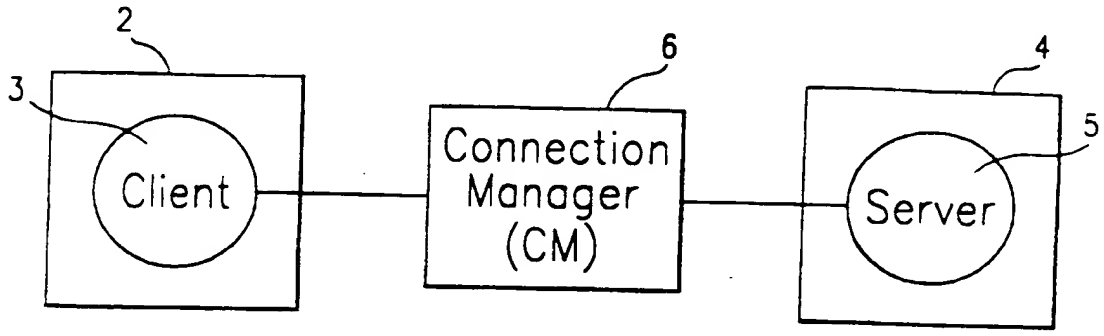


FIG. 1A

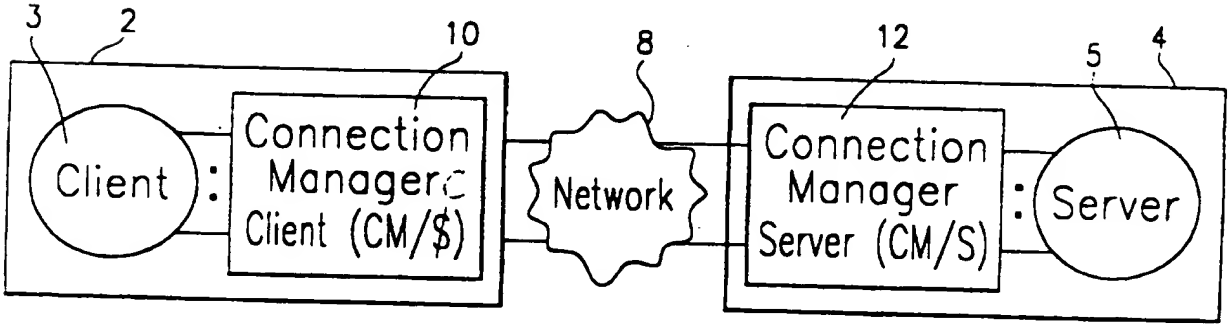


FIG. 1B

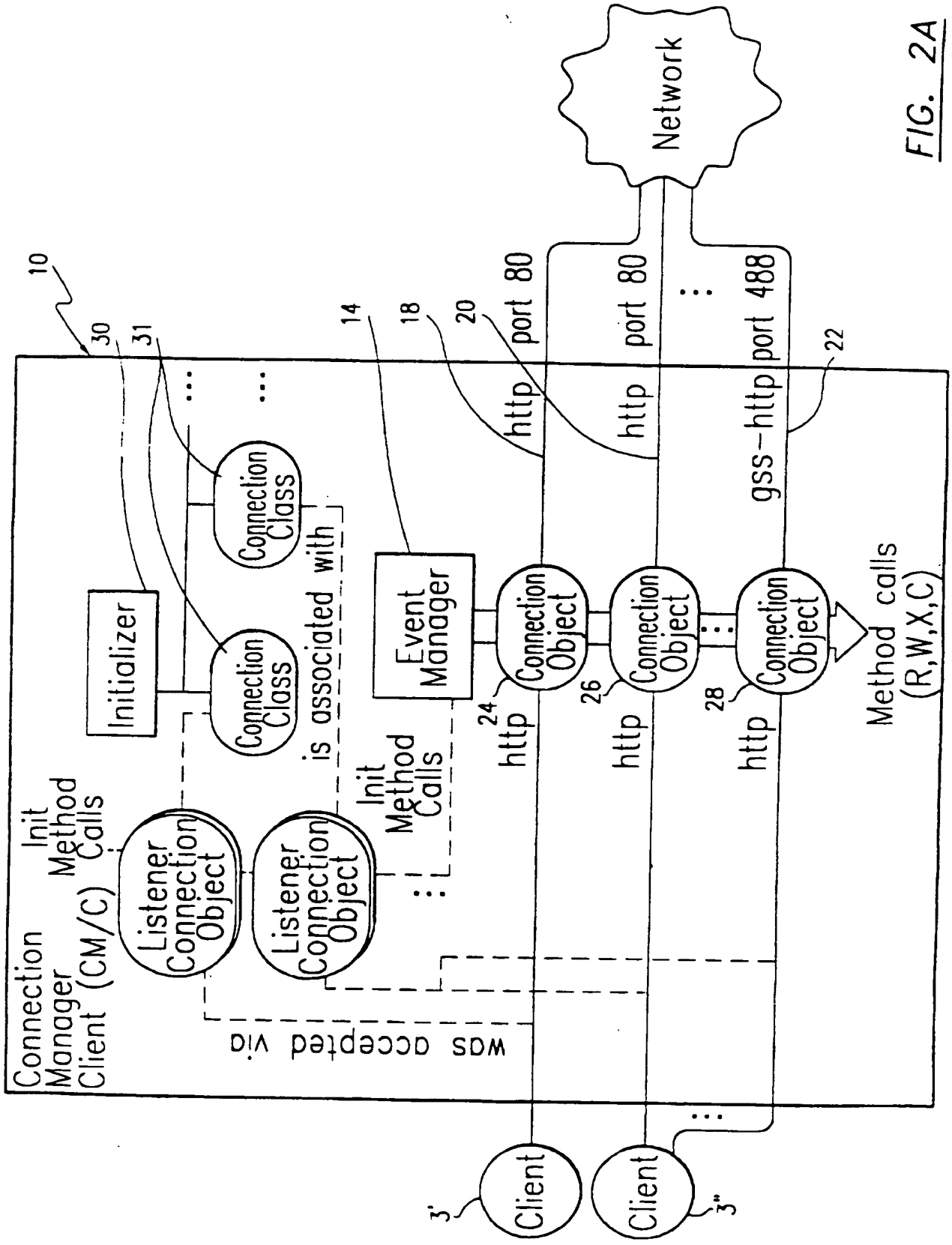
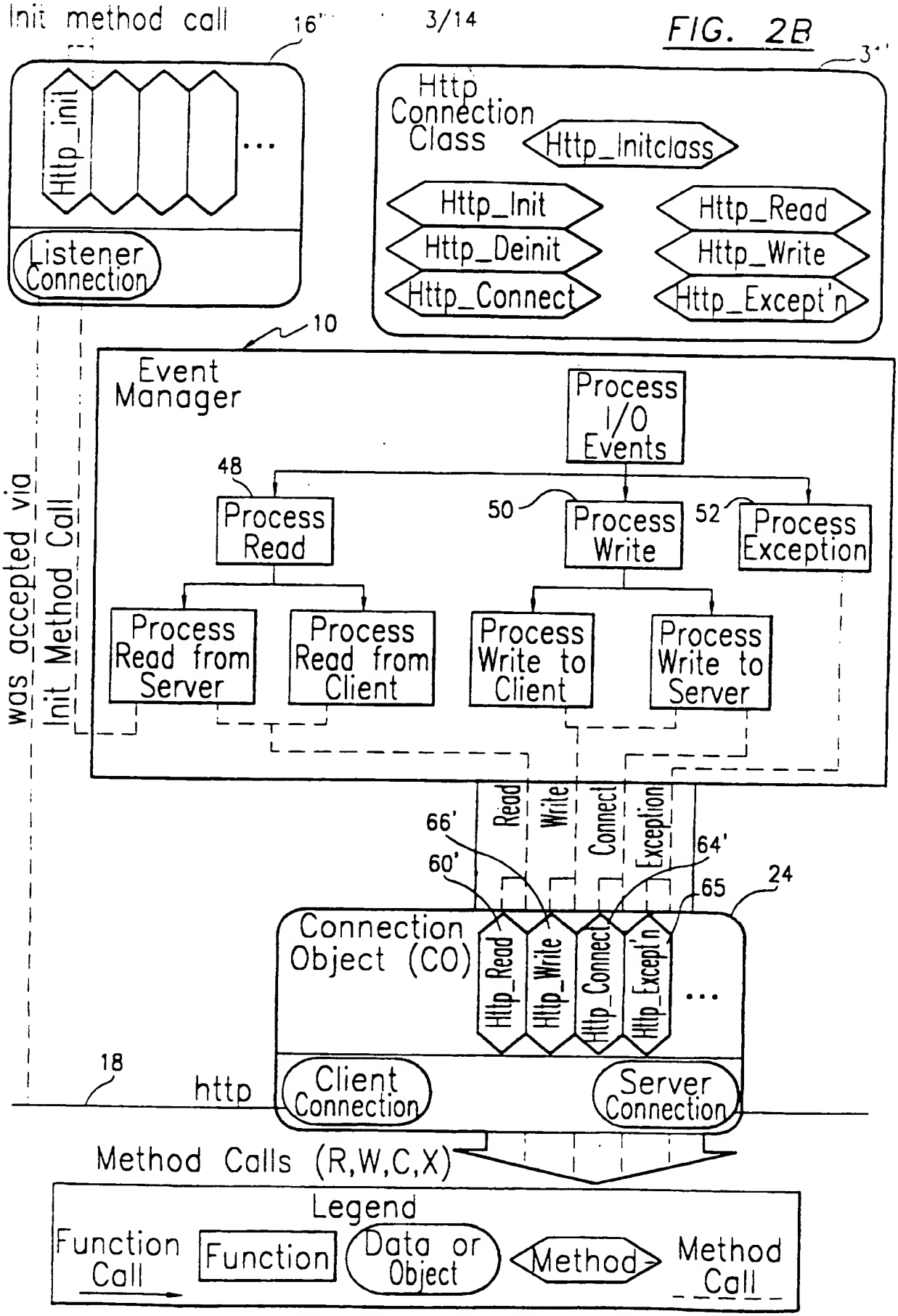


FIG. 2A

FIG. 2B



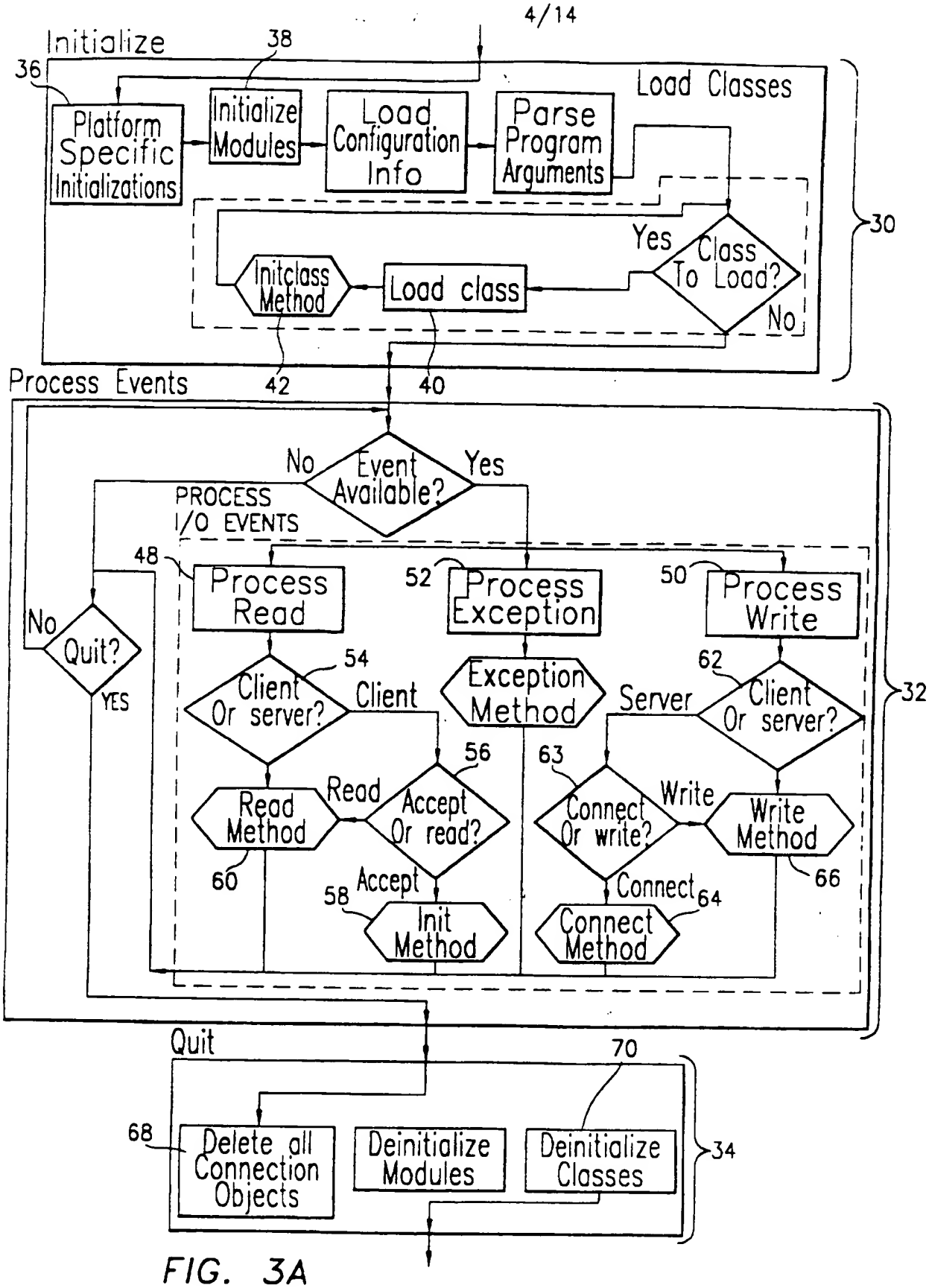


FIG. 3A

i. main(argc,argv)
  A. Initialize(argc,argv)
    1. PlatformSpecificInitializations()
    2. LoadConfigurationInfo()
    3. ModuleInitializations()
    4. ParseCommandLineArguments(argc,argv)
    5. LoadClassDefinitions()
      a)LoadClass
      b)InitClass
        (1) GenerateInitArgs
        (2) Method_ClassInit
  B. ProcessEvents()
    1. ProcessUserEvents()
    2. ProcessProgrammaticEvents()
    3. ProcessIOEvents()
      a) ProcessRead(sid)
        (1) SIDToConnectionObject(sid) ==> ConnObj,SIDType
        (2) ClientOrServerRead?
          (a) Conn_ObjProcessReadFromClient(connobj)
            i) AcceptOrRead?
              (1) ConnObj_ProcessAccept
                (a) GenerateInitArgs
                (b) Method_Init
              (2) ConnObj_ProcessRead
                (a) GenerateReadArgs
                (b) Method_Read
          (b) ConnObj_ProcessReadFromServer(connobj)
            i) Connobj_ProcessRead
              (1) GenerateReadArgs
              (2) Method_read

FIG. 3B

TO FIG. 3C

6/14 FROM FIG. 3C

(3) ConnObj_DeleteIfMarked
b) ProcessWrite(sid)
(1)SIDToConnectionObject(sid) ==> ConnObj, SIDType
(2) ClientOrServerWrite?
(a) ConnObj_ProcessWriteToClient(connobj)
i) ConnObj_ProcessWrite
(1) GenerateWriteArgs
(2) Method_Write
(b) ConnObj_ProcessWriteToServer(connobj)
i) ConnectOrWrite?
(1) ConnObj_ProcessConnect
(a) GenerateConnectArgs
(b) Method_Connect
(2) ConnObj_ProcessWrite
(a) GenerateWriteArgs
(b) Method_Write
(3) ConnObj_DeleteIfMarked
c) ProcessException(sid)
(1) SIDToConnectionObject(sid) ==> ConnOjb, SIDType
(2) ConnObj_ProcessException(connobj)
(3) ConnObj_Delete
C. Quit()

FIG. 3C

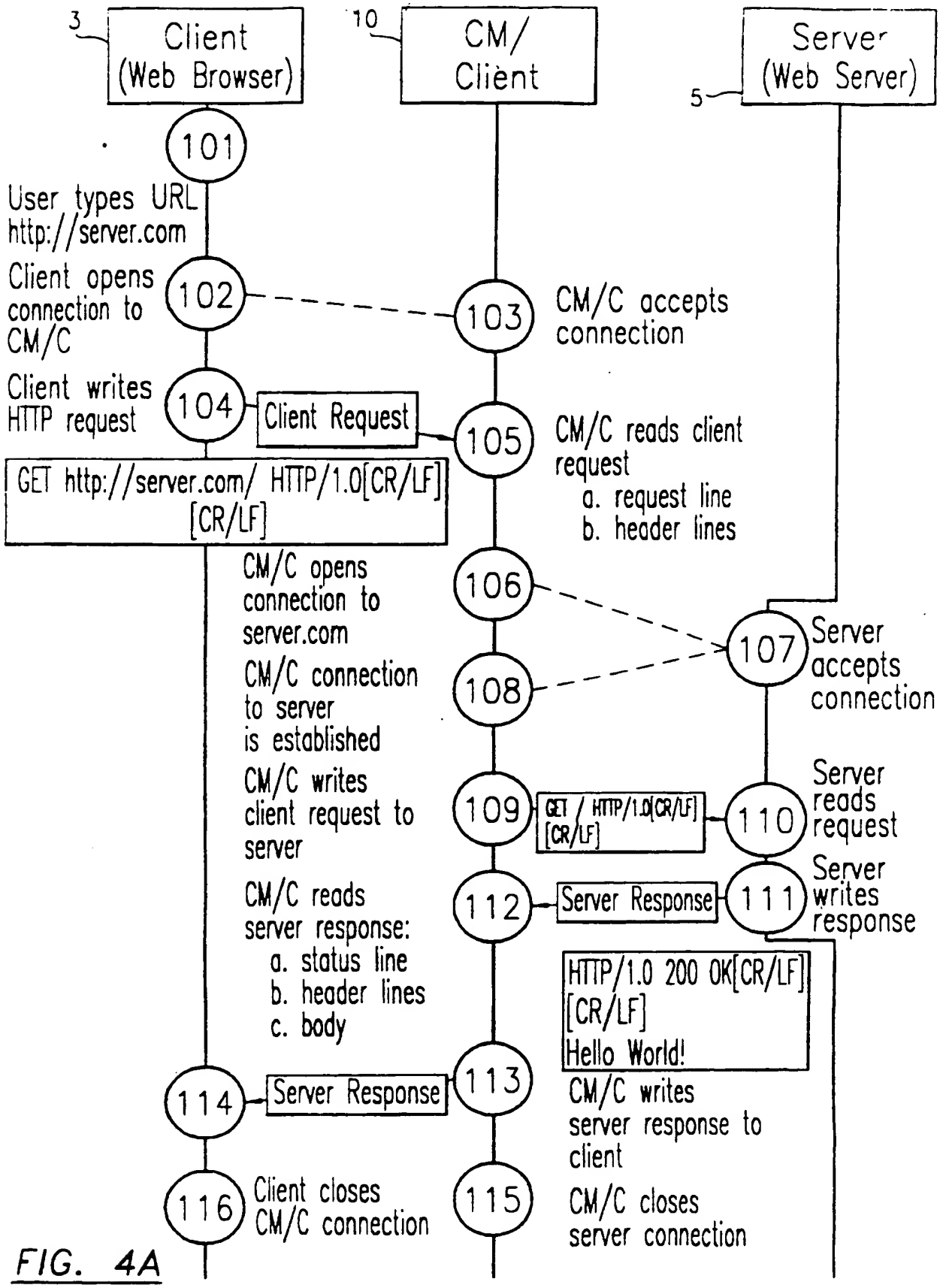


FIG. 4A

I. User types URL: http://server.com
II. Client: Open Connection to Server (in this case, CM/Client)
    A. CM/C: Accept Client Connection
III. Client: Write Request to Server (CM/C)
    A. CM/C: Read Client Request
        1. CM/C: Read Request Line
           GET http://server.com/ HTTP/1.0[CR/LF]
        2. CM/C: Read Request Headers
           [CR/LF]
        3. CM/C: Open Connection to server.com
        4. CM/C: Server Connection is Established
    B. CM/C: Write Client Request to Server
IV. Client: Read Response from Server (CM/C)
    A. CM/C: Read Server Response
        1. CM/C: Read Server Status Line
           HTTP/1.0 200 OK[CR/LF]
        2. CM/C: Read Server Headers and Body
           Content-type: text/plain[CR/LF]
           [CR/LF]
           Hello World!
    B. CM/C: Write Server Response to Client
    C. CM/C: Close Server Connection
V. Client: Read and Display Server Response
VI. Client: Close Connection to Server (CM/C)

FIG. 4B

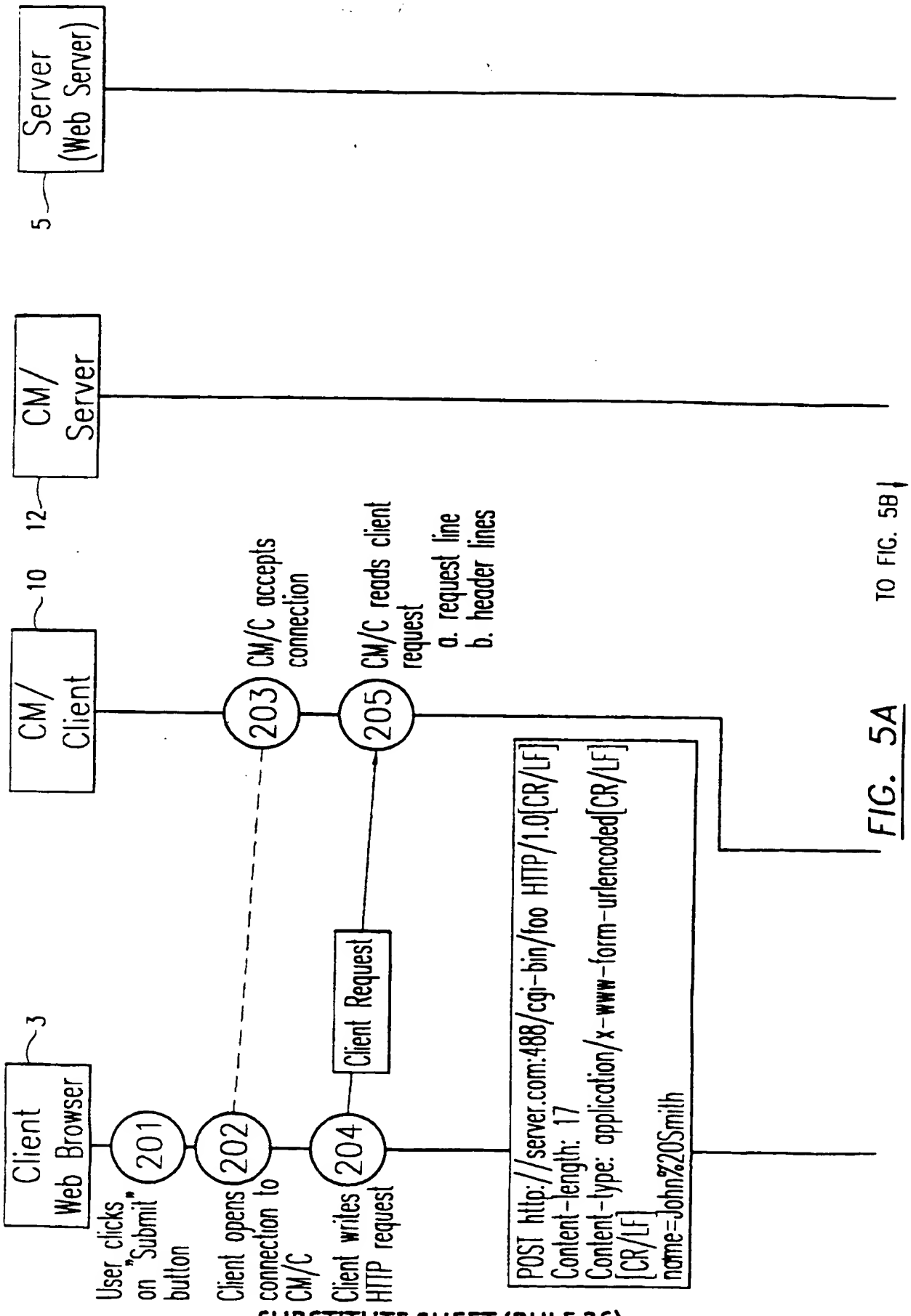


FIG. 5A

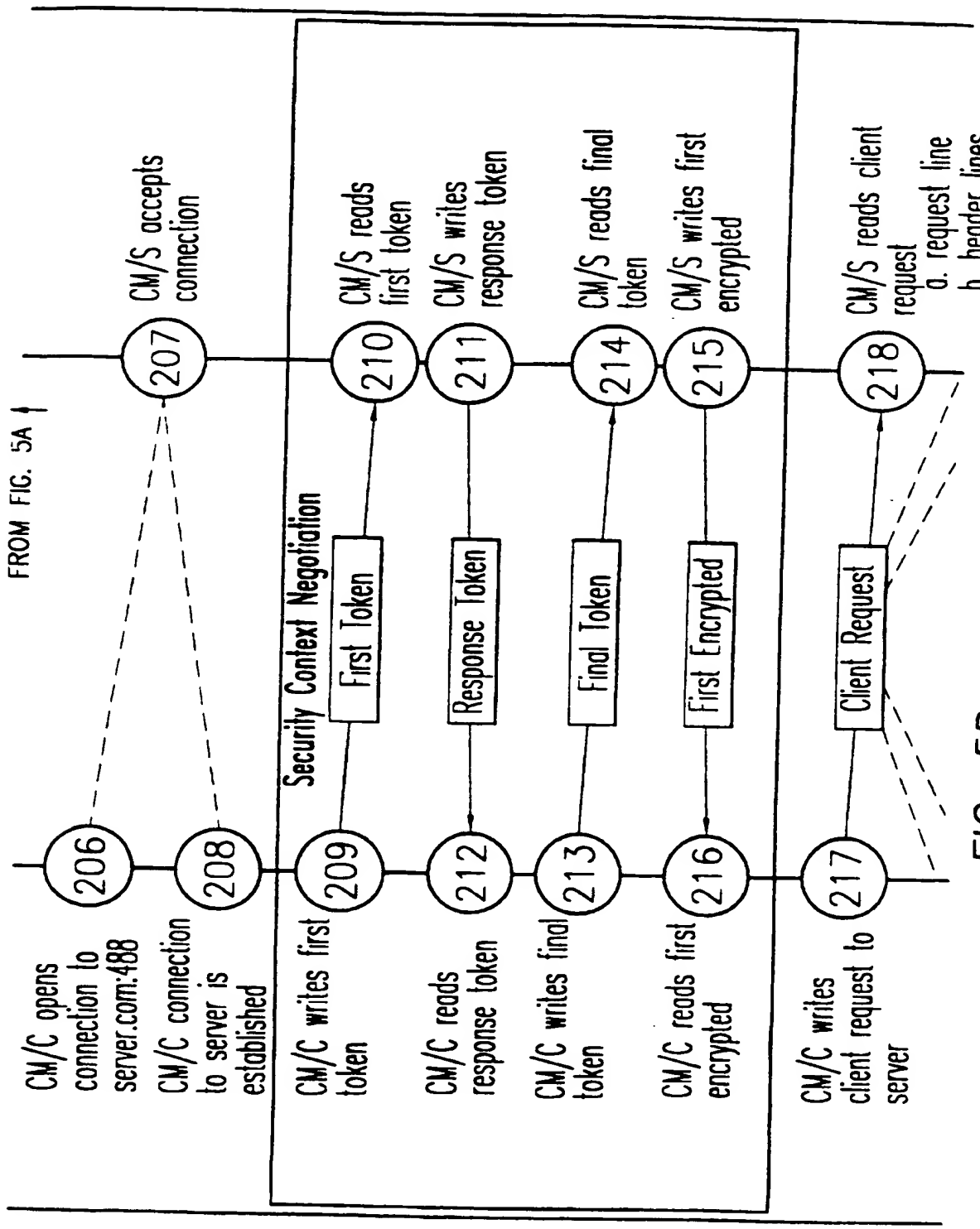


FIG. 5B

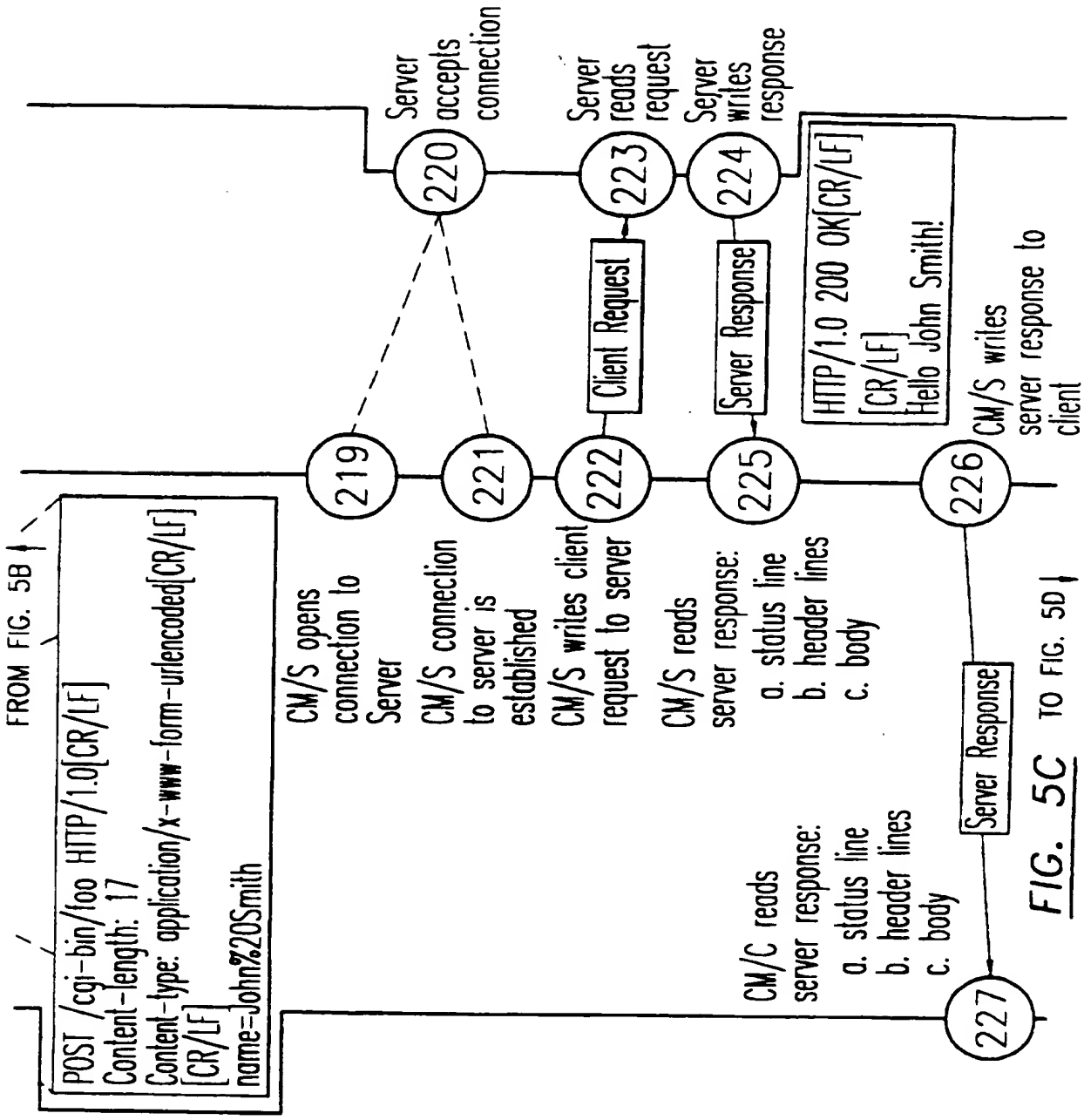


FIG. 5C

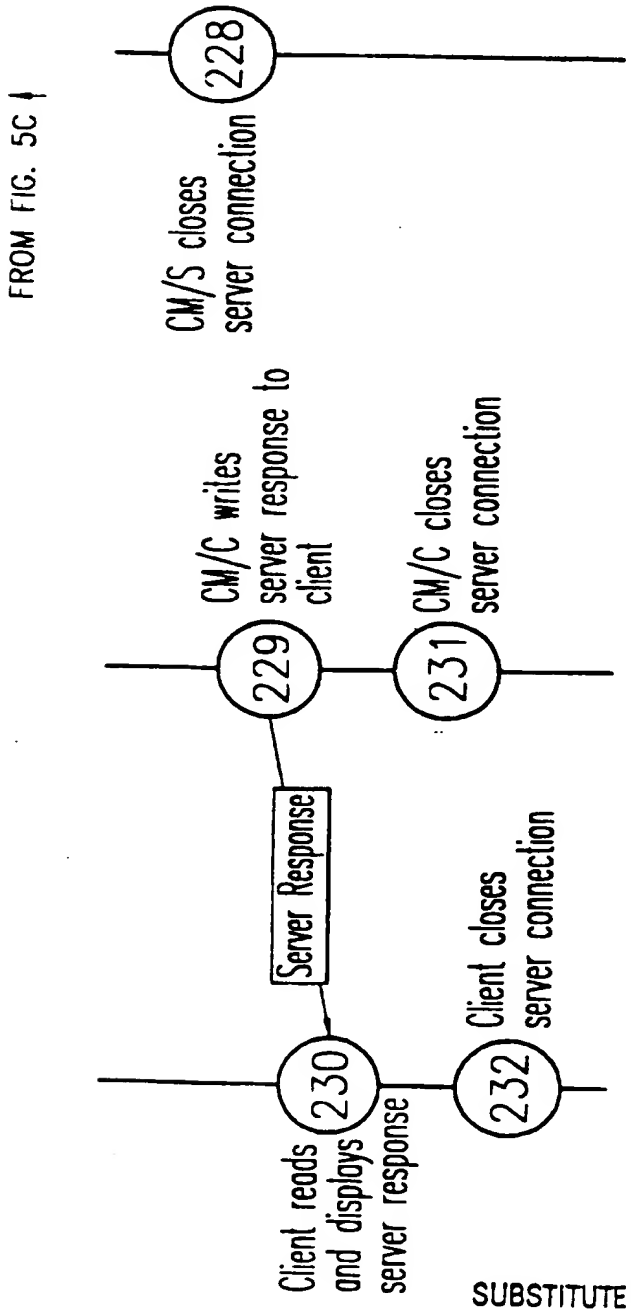


FIG. 5D

I. User clicks on "Submit" button to submit an HTML Form.
II. Client: Open Connection to Server (in this case, CM/Client)
  A. CM/C: Accept Client Connection
III. Client: Write request to Server (CM/C)
  A. CM/C: Read Client Request
    1. CM/C: Read Request Line
      POST http://server.com:488/cgi-bin/foo HTTP/1.0[CR/LF]
    2. CM/C: Read Request Headers and Body
      Accept: image/gif, image/jpeg, */*[CR/LF]
      Content-length: 17[CR/LF]
      Content-type: application/x-www-form-urlencoded[CR/LF]
      [CR/LF]
      John%20Smith
    3. CM/C: Open Connection to server.com488(in this case, CM/S)
    4. CM/S: Accept Client(in this case, CM/C) Connection
    5. CM/C: Server (CM/S) Connection is Established
  B. Negotiate Security Context
    1. CM/C: Write First Token
    2. CM/S: Read First Token
    3. CM/C: Write Response Token
    4. CM/C: Read Response Token
    5. CM/C: Write Final Token
    6. CM/S: Read Final Token
    7. CM/S: Write First Encrypted
    8. CM/C: Read First Encrypted
  C. CM/C: Write Client Request to Server (CM/S)
  D. CM/S: Read Client (CM/C) Request
    1. CM/S: Read Request Line
    2. CM/S: Read Request Headers and Body
    3. CM/S: Open Connection to Server
    4. CM/S: Server Connection is Established

FIG. 5E

TO FIG. 5F